

**Projekt z przedmiotu**

**Programowanie w języku c++**

***Temat:***

Statki

***Opracowanie:***

Marcin Wiśniowski

**Rok:** 2EF-DI

**Grupa laboratoryjna:** L02

**Data:** 13.01.2024

**Dokumentacja techniczna**

Spis treści

[Klasy i metody 4](#_Toc156085691)

1. [MainWindow 4](#_Toc156085692)
2. [MainWindow(QWidget \*parent = nullptr); ~MainWindow(); 4](#_Toc156085694)
3. [void initialize\_Labels(); void initialize\_Border(); void initialize\_Border\_GP(); void initialize\_Border\_GP\_1(); void initialize\_Buttons\_GP(); void initialize\_Buttons\_GP\_1(); 4](#_Toc156085695)
4. [void resetGame(); 4](#_Toc156085696)
5. [void cords\_for\_bot(); void gen\_Ships\_for\_players(); 4](#_Toc156085697)
6. [void shot\_for\_player(); void shot\_for\_second\_player(); 4](#_Toc156085698)
7. [void shot\_for\_bot(); 5](#_Toc156085699)
8. [vector<pair<int, int>> check\_fields(int xCord, int yCord, resources\* object); 5](#_Toc156085700)
9. [bool check\_ships(); 5](#_Toc156085701)
10. [void on\_pushButton\_Ch\_1\_clicked(); void on\_pushButton\_Ch\_2\_clicked(); 5](#_Toc156085702)
11. [void on\_pushButton\_TB\_clicked(); void on\_pushButton\_EG\_2\_clicked(); 5](#_Toc156085703)
12. [void on\_pushButton\_NS\_2\_clicked(); 5](#_Toc156085704)
13. [void on\_pushButton\_EG\_1\_clicked(); 5](#_Toc156085705)
14. [void on\_pushButton\_EG\_3\_clicked(); 5](#_Toc156085706)
15. [void on\_pushButton\_3\_clicked(); 5](#_Toc156085707)
16. [void on\_pushButton\_4\_clicked(); void on\_pushButton\_5\_clicked(); void on\_pushButton\_6\_clicked(); void on\_pushButton\_7\_clicked(); void on\_pushButton\_8\_clicked(); void on\_pushButton\_9\_clicked(); void on\_pushButton\_10\_clicked(); void on\_pushButton\_11\_clicked(); void on\_pushButton\_12\_clicked(); void on\_pushButton\_13\_clicked(); 6](#_Toc156085708)
17. [void on\_pushButton\_14\_clicked(); 6](#_Toc156085709)
18. [void on\_pushButton\_gen\_clicked(); 6](#_Toc156085710)
19. [void on\_pushButton\_res\_clicked(); 6](#_Toc156085711)
20. [Resources 6](#_Toc156085712)
21. [resources (); ~resources(); resources (const resources& other); 6](#_Toc156085713)
22. [int cordsX\_to\_numbers(const char\* x); int cordsY\_to\_numbers(int y); 6](#_Toc156085714)
23. [const char\* cordsX\_for\_Bot(int x); 6](#_Toc156085715)
24. [void set\_to\_defaults(); 6](#_Toc156085716)
25. [void set\_gamespace(const char\* x, int y, int value, int pointer); 6](#_Toc156085717)
26. [void set\_gamespace\_after\_shoot(int x, int y, int value); 6](#_Toc156085718)
27. [bool test\_correct\_positioning(int x, int y, int x1, int y1, int height); 7](#_Toc156085719)
28. [bool check\_the\_ships(int x, int y); 7](#_Toc156085720)
29. [int check\_cord\_for\_Bot(int x, int y); 7](#_Toc156085721)
30. [int check\_shot(int x, int y) const; 7](#_Toc156085722)
31. [ScoreBoard : public QObject 7](#_Toc156085723)
32. [ScoreBoard(); ~ ScoreBoard(); 7](#_Toc156085724)
33. [Int calculate\_points(int moves); 7](#_Toc156085725)
34. [void Start\_time(); void Stop\_time(); 7](#_Toc156085726)
35. [void Update\_time(); 7](#_Toc156085727)
36. [ScoreWidget 7](#_Toc156085728)
37. [ScoreWidget(QWidget \*parent = nullptr); ~ScoreWidget(); 7](#_Toc156085729)
38. [void read\_data(); 8](#_Toc156085730)

# 

# Klasy i metody

## **MainWindow**

## Najbardziej rozbudowana klasa służąca obsłudze okna, wpisywania danych przez użytkownika, grze w statki, za obsługę front-endu odpowiada technologia Qt dla języka programowania c++. Klasa ta została podzielona na 3 części, public, private oraz private slots służąca obsłudze przycisków.

### **MainWindow(QWidget \*parent = nullptr); ~MainWindow();**

Konstruktor klasy MainWindow, inicjalizuje potrzebne obiekty, zmienne oraz vectory zawierające przyciski i etykiety, ukrywa niektóre widgety, łączy bota z regulacją czasu w celu późniejszego sterowania nim, a także wywołana jest funkcja *srand(time(NULL)),* w celu zapewnienia losowych liczb. Dekonstruktor po wszystkich operacjach ustawia zmienne na wartości domyślne oraz usuwa zainicjalizowane obiekty

### **void initialize\_Labels(); void initialize\_Border(); void initialize\_Border\_GP(); void initialize\_Border\_GP\_1(); void initialize\_Buttons\_GP(); void initialize\_Buttons\_GP\_1();**

Wszystkie 6 funkcji inicjalizuje wektory zawierające przyciski lub etykiety znajdujące się w GUI całej aplikacji na różnych widżetach, w celu późniejszego dynamicznego odwoływania się do nich. Ich dołączenie do wektora następuje poprzez wyszukanie odpowiedniej nazwy obiektu znajdującego się w aplikacji. Przykładowo poszukujemy w pętli pola *Line Edit* oznaczonego nazwą *lineEdit\_i,* gdzie *i* to odpowiedni numer indeksu pola edycyjnego funkcją *findChild* oraz dopisujemy go do wektora. Dzięki takiej operacji możemy przez nazwę wektora i numer indeksu odwoływać się do odpowiednich pól. W przypadku przycisków ich przypisanie przeprowadzane jest analogicznie oraz występuje także podłączenie do nich odpowiedniego emitowanego sygnału i reakcja na niego oraz funkcja *lambda* wykonywana po jego wciśnięciu.

### **void resetGame();**

Metoda przywracająca stan początkowy dla wszystkich elementów front i back-endu. Resetuje ona tablicę na

której układane są statki poprzez wywołanie metody *„set\_to\_defaults()”,*  a także ustawia odpowiednie parametry dla przycisków używanych do wprowadzania statków, czyści pola edycyjne i zmienia arkusze stylów dla przycisków służących do gry oraz pól wyświetlających jej postęp.

### **void cords\_for\_bot(); void gen\_Ships\_for\_players();**

Metody, w którą zaimplementowany został algorytm losowego ustawiania statków, poprzez użycie pętli oraz instrukcji warunkowych oraz metod klasy *Resources* sprawdzających poprawność układania statków. Pętla iteruje od 0 do 20, przedziałami są ustawiane statki dla indeksu < 4 są to statki 1 masztowe, indeks < 7 statki 2 masztowe, indeks < 9 statki 3 masztowe indeks równy 10 statek 4 masztowy. Używając funkcji *rand % 10*, losowane są liczby z zakresu <0; 9>, w następnym kroku sprawdzane jest czy możliwe jest ich ustawienie w tablicy poprzez funkcje *check\_the\_ships* oraz *test\_correct\_positioning,* przed wykonaniem tej funkcji sprawdzany jest również wybór użytkownika czy statek 4 masztowy może mieć postać litery L. Obie funkcje zwracają logiczne wartości poprzez użycie odpowiedniej ilości flag dodawany jest statek oraz pętla przechodzi do kolejnej iteracji. Oba te algorytmy działają identycznie zarówno jak i dla użytkownika jak i dla bota, jedyną różnicą między nimi jest wypisanie w odpowiednich miejscach i pokazanie graficznie ułożenia statków dla użytkownika.

### **void shot\_for\_player(); void shot\_for\_second\_player();**

Metody wywoływane przy wciśnięciu przycisku przez gracza służące do obsługi strzału oraz sprawdzające czy aktualna gra jest przeprowadzana z komputerem czy innym graczem. Działając na odpowiednich obiektach sprawdzają czy dany strzał był w puste pole czy maszt statku. Jeżeli strzał był nietrafiony pole, w które trafiliśmy zmienia kolor na szary i wywoływana jest metoda strzału bota dla 1 wymienionej w nagłówku funkcji lub strzału dla 2 drugiego gracza dla drugiej. Przy trafionym strzale sprawdzany jest również warunek zakończenia gry oraz obliczane punkty, metodą *calculate\_points* i wypisywane w odpowiedniej klatce GUI. Sprawdzany jest również trafiony statek, jeżeli jest on trafiony i zatopiony pola wokół niego są odpowiednio kolorowane, następnie możliwe jest ponowne oddanie strzału. W obydwu przypadkach gry jedno i wieloosobowej zastosowany jest delay, w pierwszym przypadku wywołuje on funkcje strzału bota, w drugim funkcje lambda która opóźnia pokazanie GUI dla strzału drugiego gracza, oraz obydwie wyświetlają na ekran liczbę oddanych strzałów oraz aktualną turę gracza.

### **void shot\_for\_bot();**

Metoda losująca i sprawdzająca strzał dla komputera, gdy strzał jest oddany w puste pole kolorowane jest ono tak jak w poprzednim przypadku na kolor szary i odblokowywana jest możliwość strzału gracza. Gdy strzał oddany jest w maszt statku sprawdzone zostaje czy jest on cały zatopiony oraz strzał jest powtarzany, przy ograniczeniu terenu dla komputera. Gdy flaga *lastShotHit* ustawiona jest na wartość *true* komputer przeszukuje kwadrat 3x3 wokół koordynat aktualnego masztu, gdy zostanie pierwszy maszt znaleziony następuje kolejny strzał w jego koordynaty. W przypadku gdy statek nie ma innych masztów do trafienia flaga resetuje swoją wartość oraz wywołuje ponownie funkcje strzału dla bota i wartości dla strzału losowane są z całej planszy 9x9. Gdy bot ponownie trafi w wcześniej trafione miejsce strzał jest powtarzany. Sprawdzany jest także warunek zakończenia gry oraz czy trafiony statek jest w całości zatopiony, jeżeli tak, pola wokół niego są kolorowane na szaro.

### **vector<pair<int, int>> check\_fields(int xCord, int yCord, resources\* object);**

Metoda sprawdzająca czy aktualnie trafiony maszt jest ostatnim do zestrzelenia. Zwraca ona wektor par do którego są dodawane koordynaty masztów sprawdzanego statku w przypadku, gdy jest on zatopiony, algorytm sprawdza pola w pionie i poziomie, gdy zostanie napotkany niezatopiony maszt zwracany jest pusty wektor. Po sprawdzeniu pozycji sprawdzane są koordynaty pod względem ich wystąpienia w wektorze w celu wyeliminowania powtórzeń.

### **bool check\_ships();**

Metoda służąca do sprawdzenia czy wszystkie pola do wpisywania statków zostały wypełnione, wykorzystywana w celu niedoprowadzenia do powstania gracza bez ustawionym statków. Pętlą sprawdzane są wszystkie pola edycyjne, jeżeli któreś jest puste zwracana jest wartość *false* i nie możliwe jest przejście do gry w statki.

### **void on\_pushButton\_Ch\_1\_clicked(); void on\_pushButton\_Ch\_2\_clicked();**

Metody do obsługi przycisków wyboru trybu gry, zostają one wywołane po wciśnięciu odpowiedniego przycisku. Chowają odpowiednie widżety GUI, a także ustawiają odpowiedni tytuł w etykiecie, placeholdery dla pól edycyjnych, a także wartość zmiennej ukazującej wybrany tryb gry.

### **void on\_pushButton\_TB\_clicked(); void on\_pushButton\_EG\_2\_clicked();**

Metody wywołujące przeczytanie danych z pliku i wyświetlenie na ekranie aktualnej tablicy wyników.

* 1. **void on\_pushButton\_NS\_1\_clicked(); void on\_pushButton\_clicked();**

Metody wywoławany przy wciśnięciu przycisku powrotu przez gracza, wracające do wyboru trybu gry oraz resetujące wpisane nazwy gracza.

### **void on\_pushButton\_NS\_2\_clicked();**

Metoda pobierająca nazwę lub nazwy graczy w zależności od wybranego trybu gry, gdy nazwy nie zostały wpisane następuje nadanie domyślnych wartości.

### **void on\_pushButton\_EG\_1\_clicked();**

Metoda przywracająca początkowe wartości dla ustawień statków oraz pól obsługujących grę, wraca ona na sam początek i pozwala graczowi na rozgranie kolejnej rozgrywki.

### **void on\_pushButton\_EG\_3\_clicked();**

Metoda przycisku zamykającego całą aplikacje

### **void on\_pushButton\_3\_clicked();**

Metoda przycisku *GRAJ!* Tworzy ona odpowiednie obiekty w zależności od wybranego trybu gry, sprawdza czy wpisane są wartości dla masztów poszczególnych statków. W tej metodzie wywoływane jest również generowanie statków dla komputera. A także odpowiednie statki są wpisywane w etykiety na widżetach.

### **void on\_pushButton\_4\_clicked(); void on\_pushButton\_5\_clicked(); void on\_pushButton\_6\_clicked(); void on\_pushButton\_7\_clicked(); void on\_pushButton\_8\_clicked(); void on\_pushButton\_9\_clicked(); void on\_pushButton\_10\_clicked(); void on\_pushButton\_11\_clicked(); void on\_pushButton\_12\_clicked(); void on\_pushButton\_13\_clicked();**

Metody przycisków *dodaj statek* sprawdzają w pierwszej kolejności czy został poprawnie wpisany statek czy pole edycyjne nie jest puste oraz czy koordynaty są oddzielone przecinkiem. Jeżeli te dwa warunki zostały spełnione, pobierane oraz dzielone są wartości z pola edycyjnego, przekształcane są one w liczby odpowiednie do użycia jako indeksów tablicy sprawdzane są również odpowiednie flagi, takie jak możliwość wprowadzenia w danym miejscu statku oraz sprawdzenie czy dany statek może być umieszczony w tym miejscu metody: test\_correct\_positioning oraz check\_the\_ships. Jeżeli wszystkie wartości logiczne są prawdą to dodawany jest statek oraz wyświetla się on w odpowiednim miejscu na tablicy etykiet.

### **void on\_pushButton\_14\_clicked();**

Metoda przycisku *dodaj 2 gracza* sprawdza ona poprawność wpisanych wartości oraz dodaje obiekt pierwszego gracza i możliwość dodania drugiego.

### **void on\_pushButton\_gen\_clicked();**

Metoda blokująca możliwość dodania statków ręcznie oraz wywołująca generowanie i ułożenie statków przez algorytm.

### **void on\_pushButton\_res\_clicked();**

Metoda dla przycisku wywołująca funkcje *resetGame.*

## **Resources**

Klasa służąca do inicjalizacji tablicy 9x9 służącej do rozgrywki w statki oraz operacji na niej, klasa głównie składa się z publicznych metod obsługujących potrzebne zdarzenia, jedyną zmienną prywatną jest sama tablica zawierająca układ statków.

### **resources (); ~resources(); resources (const resources& other);**

Konstruktor inicjalizujący tablice z wartościami domyślnymi na 0 oraz zmienna counter również na zero. Dekonstruktor usuwający wszystkie dane z tablicy. Konstruktor kopiujący używany do utworzenia obiektów dla wszystkich graczy.

### **int cordsX\_to\_numbers(const char\* x); int cordsY\_to\_numbers(int y);**

Metody zwracające odpowiednie wartości liczbowe indeksów danych wpisywanym przez użytkownika *cordsX\_to\_numbers* zmienia litery na odpowiednie liczby dla tablic, *cordsY\_to\_numbers* pomniejsza o 1 wpisany przez użytkownika indeks Y koordynatu statku.

### **const char\* cordsX\_for\_Bot(int x);**

Metoda zwracają na podstawie wpisanej liczby odpowiednia literę.

### **void set\_to\_defaults();**

Metoda przywracająca wartości zer dla całej tablicy.

### **void set\_gamespace(const char\* x, int y, int value, int pointer);**

Metoda wpisująca statek dla obiektu w odpowiednie podane miejsce poprzez użycie metod *cordsX\_to\_numbers, cordsY\_to\_numbers* poprawiających ułożenie danych.

### **void set\_gamespace\_after\_shoot(int x, int y, int value);**

Metoda modyfikująca tablicę w zależności od wartości parametru *value* w odpowiednich indeksach *x i y*.

### **bool test\_correct\_positioning(int x, int y, int x1, int y1, int height);**

Metoda sprawdzająca ułożenie statku w pionie i poziomie w zależności od podanej wysokości. Gdy podana jest wysokość 2 statek sprawdzany jest o jedna jednostkę do tyłu, jeżeli wysokość wynosi 3 o 2 jednostki, jeżeli 4 to o 3 jednostki. W zależności czy statek spełnia zależności dla podanej wysokości zwracane są odpowiednie wartości logiczne.

### **bool check\_the\_ships(int x, int y);**

Metoda iterująca w pętli w kwadracie 3x3 od podanych wartości parametrów *x, y.* Funkcje max(…) i min(…) zostały użyte w celu zagwarantowania, że pętla nie wyjdzie poza zakres tablicy *gamespace.* Max jest ograniczeniem pętli od lewej strony, żeby nie wyszła poniżej 0, min z prawej by wartości nie wyszły poza zakres 9.

### **int check\_cord\_for\_Bot(int x, int y);**

Metoda sprawdzająca przy dodawaniu statku czy w podanych koordynatach znajduje się statek, zwraca ona 0, gdy koordynaty wskazują na puste miejsce, 1 gdy na statek.

### **int check\_shot(int x, int y) const;**

Metoda sprawdzająca czy oddany strzał o podanych koordynatach został trafiony w statek czy w puste pole oraz czy podane parametry się nie powtarzają. Metoda zwraca odpowiednie wartości w zależności od wyniku porównania.

## **ScoreBoard : public QObject**

Klasa dziedzicząca po publicznej klasie QObject. Klasa służąca do obliczenia punktów oraz zliczania czasu rozgrywki, oraz wpisywaniu wyników rozgrywki do pliku.

### **ScoreBoard(); ~ ScoreBoard();**

Konstruktor klasy inicjalizujący wszystkie zmienne oraz obiekt timer, który następnie podłączany jest do metody *Update\_time* w celu obliczania minut i sekund rozgrywki. Dekonstruktor przywracający wszystkie zmienne do stanu początkowego.

### **Int calculate\_points(int moves);**

Metoda obliczająca ilość punktów na podstawie parametru moves, następnie pobierana jest aktualna data funkcja time(nullptr) zwraca aktualny czas systemowy, przypisanie go do struktury *tm* pozwala na łatwy dostęp do daty w celu dalszych operacji. W metodzie również występują operacje wpisania danych do pliku *Data.txt* takich jak zazwa gracza obliczone punkty, liczba ruchów, aktualną datę oraz czas rozgrywki.

### **void Start\_time(); void Stop\_time();**

Metoda inicjalizująca timer, wartość ustalona na 1000 oznacza, że będzie on włączany co sekundę.

### **void Update\_time();**

Metoda aktualizująca czas co 1 sekundę, jeżeli wartość sekund osiągnie 60 to są one zerowane i wartość minut wzrasta.

### **ScoreWidget**

Klasa dziedzicząca po klase MainWindow, służy ona do wywołania nowego okna w celu pokazania tablicy wyników z poprzednich gier, zawierająca jedną funkcje pobierającą dane z pliku do zmiennej.

### **ScoreWidget(QWidget \*parent = nullptr); ~ScoreWidget();**

Konstruktor inicjuje zmienna string oraz zmienna obsługującą GUI. Dekonstruktor czyści pole tekstowe oraz wartości zmiennej *text.*

### **void read\_data();**

Metoda czytająca dane z pliku *Data.txt* w trybie do odczytu. Jeżeli zostanie on poprawnie otwarty pole tekstowe jest czyszczone oraz pętlą czytane są dane po jednej linii oraz dołączane do pola tekstowego.